Accelerating Scientific Workflows on HPC Platforms with In Situ Processing

Tu Mai Anh Do*‡, Loïc Pottier*‡, Orcun Yildiz†, Karan Vahi*, Patrycja Krawczuk*, Tom Peterka†, Ewa Deelman*
*Information Sciences Institute, University of Southern California, Marina Del Rey, CA, USA
{tudo, lpottier, vahi, krawczuk, deelman}@isi.edu
†Argonne National Laboratory, Lemont, IL, USA
{oyildiz, tpeterka}@anl.gov

2022 22nd International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 978-1-6654-9956-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CCGRID54584.2022.00009

***Abstract*—Scientific workflows drive most modern large-scale science breakthroughs by allowing scientists to define their computations as a set of jobs executed in a given order based on their data dependencies. Workflow management systems (WMSs) have become key to automating scientific workflows—executing computational jobs and orchestrating data transfers between those jobs running on complex high-performance computing (HPC) platforms. Traditionally, WMSs use files to communicate between jobs: a job writes out files that are read by other jobs. However, HPC machines face a growing gap between their storage and compute capabilities. To address that concern, the scientific community has adopted a new approach called in situ, which bypasses costly parallel filesystem I/O operations with faster in-memory or in-network communications. When using in situ approaches, communication and computations can be interleaved. In this work, we leverage the Decaf in situ dataflow framework to accelerate task-based scientific workflows managed by the Pegasus WMS, by replacing file communications with faster MPI messaging. We propose a new execution engine that uses Decaf to manage communications within a sub-workflow (i.e., set of jobs) to optimize inter-job communications. We consider two workflows in this study: (i) a synthetic workflow that benchmarks and compares file- and MPI-based communication; and (ii) a realistic bioinformatics workflow that computes mutational overlaps in the human genome. Experiments show that in situ communication can improve the bioinformatics workflow execution time by 22% to 30% compared with file communication. Our results motivate further opportunities and challenges for bridging traditional WMSs with in situ frameworks.**

***Index Terms*—Scientific Workflows, Workflow Management Systems, In situ, Pegasus, Decaf**

I. Introduction

Large-scale scientific workflows have facilitated recent major scientific discoveries [1], [2]. A scientific workflow is a complex description of computational jobs and data movements, usually represented as a directed acyclic graph (DAG) whose nodes represent compute jobs and whose edges are dependencies. Workflow management systems (WMSs) have been developed over the years [3], [4] to manage such complex sequences of computations and data movements. A WMS is responsible for scheduling different compute jobs efficiently, allocating the resources and handling data movements between storage at compute nodes or geographically distributed compute sites, like clouds and high-performance computing (HPC) centers. Traditional distributed WMSs operate at a higher granularity level than classic HPC technologies like MPI or OpenMP. A job in a workflow is often a sequential application and is seen as a black-box from the WMS' point of view. Because of that level of granularity, jobs often communicate through files.

Scientific workflows running on HPC machines have become increasingly important for large-scale science discoveries [5], [6]. The vast majority of workflows adopt a post-processing approach where a simulation produces data and writes them to disk, then several analysis kernels process these data to extract meaningful scientific insights. We have recently witnessed a dramatically growing discrepancy between the computation and I/O capabilities of HPC machines [7], [8]. A recent paradigm called *in situ* has emerged within the HPC community to overcome the inherent bottleneck arising from file-based communications—writing files to disk is several orders of magnitudes slower than using in-memory communications. Note that data-in-memory techniques are not new, IBM already proposed a similar idea with Batchpipes in their mainframe system OS/390 [9]. In situ approaches allow users to bypass costly disk accesses by leveraging faster storage layers (e.g., memory, burst buffers [10]) and high-speed interconnects. When using the in situ approach, the simulation and the analysis kernels run concurrently, and the data produced by the simulation are iteratively processed by the analysis.

Several in situ coupling frameworks have been developed in the last few years [11], [12]. However these frameworks usually lack several features offered by classic WMS, such as distributed data management, interoperability (executing jobs using different technologies within the same workflow) and the ability to run across different sites. At the same time, most of the traditional WMSs do not correctly support in situ jobs. Dorier et al. [12] listed several of these weaknesses, in particular better interoperability and data management capabilities. These conclusions indicate the need for a better integration between in situ coupling frameworks and WMSs.

In this work, we study the integration between a classic WMS, Pegasus [3], and the Decaf [13], an in situ middleware. Decaf allows us to benefit from in-memory data transfers between jobs while Pegasus WMS relies on file communi-

‡Equal contribution

978-1-6654-9956-9/22/$31.00 ©2022 IEEE
DOI 10.1109/CCGrid54584.2022.00009

cations. Note that, as Decaf relies on MPI, we exclusively consider high-performance computing platforms in this work. Pegasus includes a number of workflow execution engines that target specific execution environments. One of the engines is designed to run on HPC platforms and executes a sub-workflow using MPI and a master/worker approach (i.e., each job within the sub-workflow has a MPI rank and an extra MPI rank is the master that organizes communications among ranks). We propose a new Decaf-based execution engine that uses Decaf to manage communications among a set of jobs within a sub-workflow. Then, we compare the performance obtained via Decaf engine with baseline performance using native Pegasus MPI-based engine.

This paper makes the following contributions:

1) We propose a new workflow execution engine in Pegasus WMS that allows users to leverage Decaf in situ framework.
2) Armed with two workflows, a simple synthetic workflow which only performs I/Os and a realistic bioinformatics workflow that computes human genome mutations, we demonstrate the utility of such integration between an in situ framework and a classic task-based WMS, and we quantify the performance gains using a leadership-class HPC machine.
3) We highlight the lessons learned, and discuss how task-based WMSs could address deeper integration of emerging computing paradigms such as in situ processing.

This paper is organized as follows. In Section II, we review important related works about task-based WMS and in situ synergies. Section III introduces Pegasus, Decaf, and thoroughly explains the proposed integration. Section IV presents an I/O synthetic workflow that we use to explore potential performance benefits arising from leveraging in situ communications; then we introduce a data-intensive bioinformatics workflow that helps us validate the proposed approach on a realistic use-case. In Section V, we evaluate our approach using the two aforementioned workflows; we summarize our findings, and we discuss future challenges and opportunities that in situ brings to established task-based WMS. Section VI concludes our work and presents potential future work directions.

II. Related Work

A. From Loosely to Tightly-coupled Workflows

Traditionally, a workflow is represented as a directed acyclic graph (DAG), where computational jobs are the nodes and the edges represent data and control dependencies between those jobs. Most of current WMSs have adopted a loosely-coupled approach, in which jobs communicate through files – a job writes output files that are then used as input files for another job. However, this model does not facilitate inter-job communications during their executions, which are the heart of tightly-coupled approaches [14]. Existing, well-established workflow management systems have been designed before the emergence of tightly-coupled workflows [15], [16]. Bringing tightly-coupled workflow support into current and emerging WMS is one key element on the road to enable computational science at an extreme-scale [7], [16]. Some traditional DAG representations have been extended with the concept of a *bundle* representing a group of several jobs that need to be scheduled concurrently, allowing inter-jobs communications at runtime [3]. Our work provides another approach to transform loosely-coupled workflows (e.g., HTC) to tightly-coupled workflows (e.g., HPC).

B. In situ Workflows

Unlike traditional WMSs, which rely on file systems to exchange data, in situ workflows run within a single HPC system, and data exchange is done through memory or the supercomputer interconnect. For example, VisIt's Libsim [17] and Paraview's Catalyst [18] libraries support shared-memory communication between analysis and visualization tasks running synchronously with the simulation, in the same address space. Another example is Damaris [19], I/O middleware that supports in situ data processing and visualization using dedicated cores, where messages between simulation and visualization tasks are allocated in a shared-memory buffer. Decaf [13] is a middleware for coupling parallel tasks in situ by creating communication channels over HPC interconnects through MPI. Some in situ solutions offload the data to a distributed memory space that is shared among multiple workflow tasks. DataSpaces [20] provides such a distributed memory space, where workflow tasks can both push data into this space and retrieve data from it. Another example is FlexPath [21], which provides a publish/subscribe model to exchange data between parallel codes running on separate resources. In this work, we explore using in situ frameworks in task-based WMSs to accelerate scientific workflows.

The idea of combining the best of both worlds is not new. Yildiz et al. [22] studied the execution of heterogeneous workflows with traditional task-based workflow tasks and in situ tasks, where PyCOMPSs [23] manages the end-to-end workflow, and Decaf [13] is used as an in situ middleware. The resulting heterogeneous workflow exhibited speedup increases over a traditional task-based workflow. While [22] was one of the first steps exploring the potential benefits of heterogeneous workflows by integrating task-based and in situ workflows, our work focuses on the performance benefits brought by using in situ solutions in traditional task-based workflows, and provides a deeper analysis for such benefits with respect to different workflow characteristics.

III. Integration

In this section, we describe the main contribution of this work, the integration between the Pegasus workflow management system and the Decaf in situ framework. We first introduce Pegasus and Decaf, then we describe our proposed integration and how it can accelerate scientific workflows. Figure 1 presents an overview of the integration that we propose in this work from a user perspective.

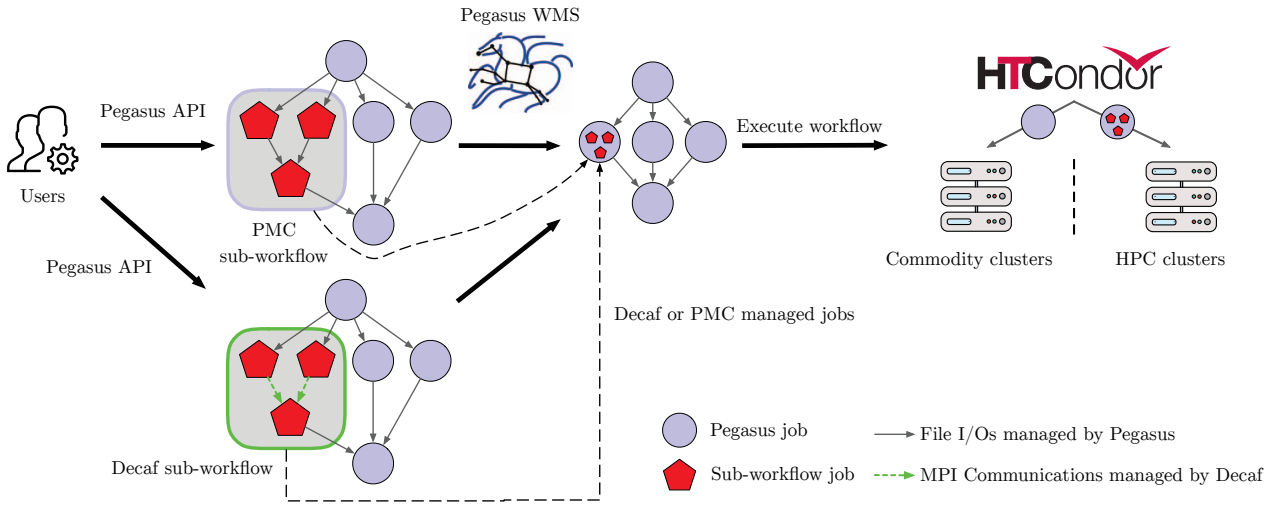


Figure 1: Users express their computations using the Pegasus API and have the possibility to execute portions of the workflow (or the whole workflow) using Pegasus with the *pegasus-mpi-cluster* (PMC) engine. Then, Pegasus creates a workflow that is submitted to the underlying scheduler (e.g., HTCondor, Slurm). In this work, we give users the possibility to leverage an in situ framework, Decaf, when executing sub-workflows. From the workflow structure, Pegasus automatically infers the correct Decaf representation and creates the appropriate workflow representation, which is then submitted to a HPC scheduler. Thus, users can easily leverage this feature in their existing Pegasus workflows by simply annotating the jobs that have to use Decaf.

A. *Pegasus Workflow Management System*

Pegasus [3] is a well-established workflow management system that enables users to describe scientific workflows, where the descriptions are independent of the available resources to execute the workflow tasks, and they are also independent of the location of data and executables. Data transfer tasks are automatically added to the executable workflow and they perform two key functions: (1) stage in input files to staging areas associated with the computing resources, and (2) transfer the generated outputs back to a user-specified location. Additionally, data cleanup (removal of data that is no longer required at the execution site) and data registration tasks (cataloging the output files) are also added to the workflow. The main workflow execution engine in Pegasus is HTCondor's *DAGMan* and individual jobs are executed by HTCondor's *schedd* [24]. HTCondor is a job management system particularly well suited for distributed high-throughput computing (HTC) environments, to run and manage the generated workflows.

B. *Job Clustering*

An important feature that Pegasus, as well as other WMSs, offers is *job clustering* [25] where multiple jobs are grouped and executed together as one larger single unit. Job clustering is a widely-used technique to increase throughput and improve workflows' performance by improving data locality (i.e., traditional heuristic is to cluster jobs sharing common data). Job clustering also greatly benefits workflow runtime by reducing the number of jobs in the queue, thus the overall queuing time incurred.

Pegasus supports two job clustering flavors, an automatic mode where Pegasus identifies and clusters independent jobs together based on predefined characteristics (e.g., clustered job should not run longer than a preset runtime), and a manual mode, where users directly label jobs they want to cluster together. In this study, we extend the latter to in situ.

By definition, a clustered job contains a sub-workflow of the original workflow. Pegasus provides different ways of executing this sub-workflow. In the basic case, the jobs within the sub-workflow are executed using an engine called *pegasus-cluster* that executes the jobs sequentially, following a topological sort of the sub-workflow. Pegasus also supports execution of jobs within the sub-workflow using an engine called *pegasus-mpi-cluster* (PMC) [26] that leverages the message passing interface (MPI) [27] to execute the jobs using a master/worker paradigm (i.e., each job within a given sub-workflow is managed as an MPI rank, and PMC creates an additional MPI rank for the master). Note that both workflow engines rely on files to handle data exchanges between jobs of the same sub-workflow, and even though PMC leverages MPI to manage sub-workflow jobs, every I/O operation between jobs uses files. Unfortunately, file-based I/Os are slow, especially on HPC platforms, where file systems (e.g., Lustre) are usually distributed over several I/O nodes. In addition, PMC cannot cluster jobs that are already using MPI. In this work, we propose a novel workflow execution engine in Pegasus based on the Decaf [13] in situ framework, allowing us to leverage faster in situ communications.

C. In Situ Communications with Decaf

Decaf [13] is a middleware for building and executing in situ workflows. Decaf allows parallel communication of coupled tasks by creating communication channels over HPC interconnects through MPI. In particular, Decaf creates parallel communication channels, each association of a producer, a consumer, and a communication object to exchange data between the producer and the consumer over MPI. Producers and consumers are parallel programs, each with their own MPI communicator. Decaf creates an additional communicator between the producer and the consumer for the data exchange. Messages are passed in a distributed fashion, point-to-point from producer ranks to consumer ranks, without aggregating at the root of either the producer or the consumer. Decaf launches these parallel programs as a multiple-program-multiple-data (MPMD) MPI application.

D. In situ Workflow Execution Engine

The goal of this work is to accelerate existing scientific workflows executed via Pegasus WMS with the use of an in situ workflow execution engine (i.e., rely on Decaf to manage inter-jobs I/Os within a specific sub-workflow). Figure 1 illustrates our proposed solution. The existing PMC route is represented on the top-region of the figure, where users can execute sub-workflows using PMC by labeling the jobs, and the lower part of the figure highlights our contribution. Users can now use the Decaf in situ framework to manage the execution of these sub-workflows. Users have to only indicate to Pegasus where to find Decaf library, and then label the jobs they want to place in a sub-workflow using identical labels and Pegasus will automatically infer the correct Decaf representation for the sub-workflow. In addition, user codes require some changes to use Decaf primitives in order to replace file-based I/Os with MPI communications. In particular, to minimize the required code modifications to the user codes, Decaf provides a simple put/get API [28] that allows tasks to send/receive data to/from the rest of the workflow. Our proposed solution allows users to benefit from faster data exchanges when running on a HPC cluster with MPI installed while only requiring minimal changes for the existing workflows.

IV. Workflow Use Cases

In this section, we present two workflow use cases. The first one is a synthetic I/O-intensive workflow, which allows us to explore simple scenarios and highlight in which cases Decaf integration brings performance improvements. The second workflow is a realistic data-intensive bioinformatics workflow, 1000 Genomes, that computes human genome mutation overlaps. Both workflows are available online [29], [30] as open-source projects for interested readers to reproduce our findings.

A. I/O Synthetic Workflow

In order to investigate and understand early potential benefits of leveraging the in situ workflow engine in Pegasus, we designed a simple chain workflow called SyntheticIO (see Figure 2). SyntheticIO has five I/O jobs, each of them reading and writing a file of a given size ranging from 1 to 16 GB. In addition, by default each job in SyntheticIO is

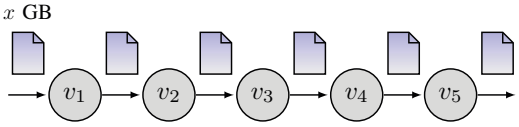


Figure 2: SyntheticIO with 5 jobs, each job reads/writes a file of x GB ($x \in \{1, 2, 4, 8, 16\}$). In addition, each job sleeps for 2 seconds per GB written (e.g, if v_2 writes 2 GB then sleep time is equal to 2×2 seconds)

purely doing I/O and does not do any computation. However, we can configure the workflow such that each I/O job sleeps for 2 seconds per GB of data written. The idea behind this behavior is to emulate a computation in order to study how the Decaf engine can help us overlap communications with computations.

B. 1000 Genomes Workflow

The 1000 Genomes workflow (shortened as Genome in this study) is a bioinformatics workflow that fetches, parses and analyzes data from the 1000 Genomes Project [31], which aims to provide a comprehensive reference of human genetic variations. The workflow analyzes mutational overlaps in

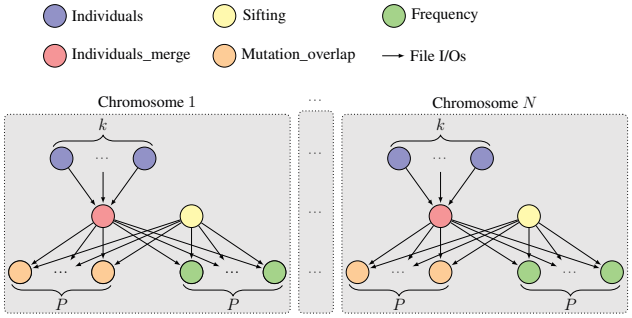


Figure 3: Genome [30] workflow with N chromosomes and k *Individuals* jobs per chromosome and P super populations ($N = 1$ and $P = 7$ in this study).

humans, ultimately allowing statistical evaluation of potential disease-related mutations.

The Project's Phase 3 and superpopulations data are downloaded and parsed (*Individuals* and *Individuals_Merge* jobs), sorting amino acid substitutions and determining their potential phenotypic effects (*Sifting* jobs). Then, several analysis tasks are performed in *Frequency* and *Mutation_Overlap* jobs, and lastly an output dataset is created (see Figure 3).

Workflow Characteristics. Genome is a data-intensive workflow that processes a given number of chromosomes in parallel (from 1 to N), where a chromosome file is quite large with $250,000$ lines, hence, the workflow leverages data parallelism to process $250,000/k$ in parallel. In addition, this workflow

relies on 26 populations, but uses seven super populations: African (AFR), Mixed American (AMR), East Asian (EAS), European (EUR), British from England and Scotland (GBR), South Asian (SAS), and all the populations together (ALL). Thus, the workflow has 7 *Frequency* jobs and 7 *Mutation_Overlap* jobs per chromosome processed. Therefore, a given execution of GENOME has $N \times (k + 2 + 2P)$ jobs in total.

Performance characterizations. In this work, we present a new method to execute Pegasus sub-workflows using the Decaf in situ framework. Once we have the scientific description of the workflow, we have to define which jobs should be executed by which workflow engines (usually based on the target execution platform). To this end, we study the execution time breakdown within one run of GENOME and the I/O characteristics of *Individuals* jobs.

Job	Execution Time (s)	Fraction (%)
Individuals	11,431	81.85
Frequency	1,492	10.68
Individuals_Merge	500	3.58
Mutation_Overlap	468	3.35
Stage_Out	34	0.24
Stage_In	21	0.15
Auxiliary[1]	16	0.11
Sifting	6	0.05
Total[2]	($\approx$ 3.9h) 13,967	100

[1] Internal jobs managed by Pegasus.
[2] Total execution time is not the makespan of the workflow, it is simply the sum of all job execution times (i.e., no notion of scheduling).

Table I: Execution time breakdown within GENOME with $k = 10$ *individuals* jobs and 1 chromosome. This instance has been executed using Cori at NERSC (see Section V-A for further details about the platform).

Table I presents the execution time for each job and their relative time fraction of the entire workflow. When there are multiple jobs from the same class (e.g., *Individuals*, *Mutation_Overlap*, and *Frequency*) that could run in parallel, we take the maximum duration among those jobs. For example, we take the maximum execution time among the 10 *Individuals* job executions. Our goal here is not to focus on the raw execution time, but rather on the relative time of each job. We observe that *Individuals* clearly dominates the workflow execution time with more than 80% of total time spent in *Individuals* jobs. Based on that observation, we conjecture that clustering k *Individuals* jobs and one *Individuals_Merge* job together and executing them with a specialized in situ execution engine can improve performance.

In order to back up this conjecture, we explore I/O characteristics of *Individuals* jobs in Table II. Note that each *Individuals* processes a part of the chromosome file, then compresses the $2,504$ resulting files into a single archive file, and sends this resulting archive to *Individuals_Merge*. On the other hand, *Individuals_Merge* is in charge of decompressing k archives and merging all k partial results. In Table II, we observe that *Individuals* jobs exhibit a large memory footprint and perform a substantial amount of file I/O by writing many files ($2,504 \times k$) to the filesystem. Such I/O patterns could greatly benefit from in-memory communications between these jobs. Based on these conclusions, in the rest of this study, we consider exclusively a subset of GENOME comprised of k *Individuals* jobs and one *Individuals_Merge* job (see cluster in Figure 4) as these jobs represent more than 85% of the workflow execution time.

V. EXPERIMENTS

A. Experimental Setup

Platform. Our execution platform is Cori [32], a Cray XC40 supercomputer located at the National Energy Research Scientific Computing Center (NERSC). Each compute node is equipped with two Intel Xeon E5-2698 v3 (16 cores each) sharing 128 GB of DRAM. Nodes are connected to each other by a Cray Aries interconnection network. In this work, we use exclusively CPU nodes to run each of the workflows.

Software. For all experiments in this work, we use Pegasus WMS 5.0.2dev [33] (commit ID: `9bb674f6f`), Decaf [34] (commit ID: `ad6ad82`), and HTCondor 8.8.1 [24]. The I/O synthetic workflow [29] and 1000 Genomes workflow [30] are available online as open-source projects with complete documentation allowing interested users to reproduce our findings.

B. Execution Scenarios

We have two use cases serving different purposes: (*i*) SYNTHETICIO is a synthetic workflow used to explore potential gains from using Decaf, and (*ii*) a realistic bioinformatics workflow, GENOME, used to validate our approach. We define three execution scenarios and describe how we compute the workflow execution time for each scenario. Note that for GENOME, we only consider k *Individuals* jobs and one *Individuals_Merge* job, not the entire workflow. Table III summarizes different scenarios and the deployment used for each scenario and use case.

VANILLA. The first scenario acts as a baseline scenario, where we run the use cases with Pegasus default settings, using the default execution workflow engine, which relies on file-based communications. There is no sub-workflow in this case. Every task is submitted to the Slurm [35] scheduler as a single job by Pegasus, and each job is executed on one compute node since jobs are scheduled on the regular queue of Cori [32]. In particular, each job runs on a dedicated one-node allocation to avoid potential interference.

As each job is submitted as a standalone job, concurrent jobs (e.g., *Individuals*) can run at different times depending on the queue status. Therefore, measuring the wall time without the time spent waiting in the queue is not straightforward, and we compute the workflow makespan based on the execution time recorded for each job. For SYNTHETICIO, we sum the execution times of each job in the chain of jobs. For GENOME,

# of *Individuals* (k)	Input per *Individuals* (lines)	# of output files per *Individuals*	Output size per *Individuals* (MB)	Peak Mem. per *Individuals* (GB)
2	125,000	2,504	92.66 (±1.88e-04)	6.11 (±1.76e-05)
5	50,000	2,504	39.43 (±2.28e-04)	3.95 (±7.94e-06)
10	25,000	2,504	21.19 (±9.90e-04)	3.25 (±7.94e-06)
16	15,625	2,504	10.33 (±1.41e-04)	2.93 (±1.49e-04)

Table II: I/O characteristics of *Individuals* jobs in GENOME. Each value is the result of 3 trials.

	VANILLA	PMC	PEGDECAF
Job scheduling	One allocation per job	Single allocation	Single allocation
SYNTHETICIO Workflow			
Number of nodes	5	1	5
Makespan	$\sum_i \mathcal{T}(v_i)$	Measured wall time	$\max_i (\mathcal{T}(v_i))$
GENOME Workflow			
Number of nodes	Number of *Individuals* + 1	Number of *Individuals*	Number of *Individuals* + 1
Makespan	$\max_k (\mathcal{T}(\textit{Individuals}_k)) + \mathcal{T}(\textit{Individuals\_Merge})$	Measured wall time	$\max_k(\mathcal{T}(\textit{Individuals}_k), \mathcal{T}(\textit{Individuals\_Merge}))$

Table III: Deployment settings for each scenario and use case, i denotes the number of I/O jobs and k denotes the number of *Individuals* jobs. $\mathcal{T}(\cdot)$ denotes the job execution time measured by Pegasus.

we take the longest execution time among the k concurrent *Individuals* jobs, and we add this time to the execution time of *Individuals_Merge*, as we only consider k *Individuals* jobs and one *Individuals_Merge* job.

PMC. In this second scenario, we leverage the *Pegasus-MPI-Cluster* [26] engine to execute portions of the workflow. Note that the user-defined sub-workflow (containing multiple jobs clustered together) appears to Pegasus as a single job, and it is submitted to Slurm as a single allocation. Therefore, instead of submitting several jobs to Slurm as in the baseline scenario (VANILLA), Pegasus submits only one job. Moreover, like VANILLA, each job in the sub-workflow runs on a single node to avoid potential interference. Based on the level of parallelism (i.e., how many jobs could run at the same time), we allocate sufficient number of compute nodes to the allocation such that (*i*) each job in the sub-worklfow runs on a single dedicated node and (*ii*) within the sub-workflow jobs having no data dependency can run simultaneously (e.g., for a sub-workflow containing 5 concurrent jobs, we allocate 5 different compute nodes; for a sub-workflow of 5 sequential jobs, we only need to allocate 1 compute node).

In the case of SYNTHETICIO, all the jobs are clustered together into one sub-workflow, while we cluster all *Individuals* and the *Individuals_Merge* jobs together for GENOME, as shown in Figure 4 based on our previous workflow characterization study. We schedule an allocation for the workflow with a certain number of compute nodes such that the allocation can sustain the largest number of concurrent jobs required by the workflow tasks' dependencies. For the SYNTHETICIO, all I/O jobs are performed sequentially, hence, we only need a single-node allocation, and PMC will schedule the jobs one after the other. For the GENOME, we assign an allocation with the number of nodes that is equal to the number of *Individuals* so that they are able to execute concurrently. We use the wall time recorded by Pegasus to obtain the makespan of the workflow execution.

PEGDECAF. In this third scenario, we aim to bring benefits of in-memory data transfers managed by Decaf to Pegasus. This scenario is similar to PMC, but instead of reading and writing files to communicate between jobs, the workflow engine enables communications via MPI calls as shown in Figure 4.

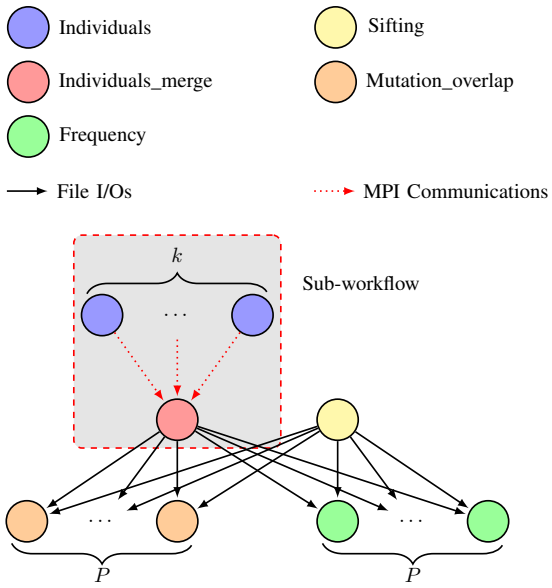


Figure 4: GENOME with $N = 1$ chromosome when clustering *Individuals* and *Individuals_Merge* jobs into a sub-workflow. When using the Decaf engine the jobs within the sub-workflow communicate via MPI, with PMC, they communicate via files.

Specifically, Pegasus coordinates the job execution, while data communication between jobs that belong to a sub-

workflow is managed by Decaf. Decaf uses MPI instead of traditional file communication and maintains communicators among different jobs so that jobs with data dependencies can exchange data through MPI operations on data residing in memory. Note that this model requires jobs within a sub-workflow to run simultaneously, i.e., they start at the same time. Therefore, this scenario requires sufficient resources to accommodate all clustered jobs to run concurrently. For both workflows, because all jobs start at the same time, the makespan is simply the execution time of the longest running job (see Table III).

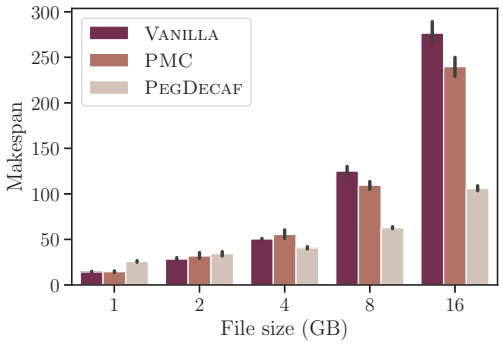
C. Results

In this section, we study the performance of both use cases, SYNTHETICIO and GENOME, using the three scenarios described previously.

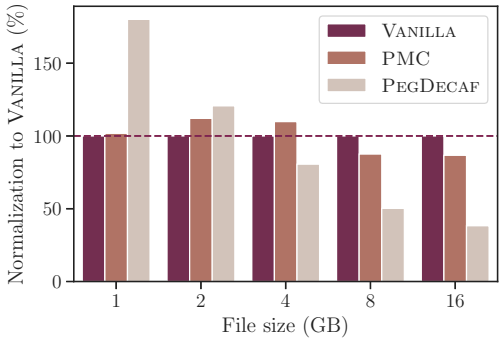
SYNTHETICIO. We conduct the experiments by running the SYNTHETICIO workflow of 5 I/O jobs, in which the size of data read and written by each job is varied between 1 GB and 16 GB. This workflow has 2 modes: without and with the sleep, i.e. the I/O jobs sleep for a certain amount of time after data is written (2 seconds per GB written). As further detailed in Section IV-A, the intuition behind this sleep time is to highlight Decaf capabilities of interleaving computations with communications among concurrent jobs. Figures 5 and 6 display SYNTHETICIO makespan, respectively, with and without the sleep time. The makespan is measured as described in Table III, and all data points are averaged over 10 runs.

In the no sleep scenario presented in Figure 5, VANILLA and PMC scenarios show similar performance. That result is expected as they both use files to manage data transfers between jobs. However, because the jobs in the VANILLA scenario are possibly scheduled at different times, the execution time of each I/O job slightly varies depending on the contention of the shared parallel file system. On the PEGDECAF side, results are promising with our in situ-based worfklow engine demonstrating the lowest makespan, especially when operating on files larger than 4 GB. However, PEGDECAF performs worse at small file sizes, which is due to the overhead of MPI calls that Decaf uses for in situ communications, with such small file sizes these calls cannot be amortized. This overhead diminishes when the file size increases. For instance, in the case without sleep, PEGDECAF starts to outperform the baseline VANILLA at 4GB and improves the makespan up to 62% at 16 GB.

With the sleep mode enabled on Figure 6, PEGDECAF outperforms the other two scenarios when using 2 GB and larger file sizes. In particular, PEGDECAF improves makespan up to 71% compared with the VANILLA configuration at 16 GB. Comparing the no sleep configuration with the sleep one, we see that PEGDECAF brings more benefits when sleep is enabled. This is due to the overlapping in situ communications with computations (e.g., as emulated by sleeping in this use case). In classic file-based WMS, a job starts its execution only when its parent jobs finish their execution and generate their



(a) Makespan



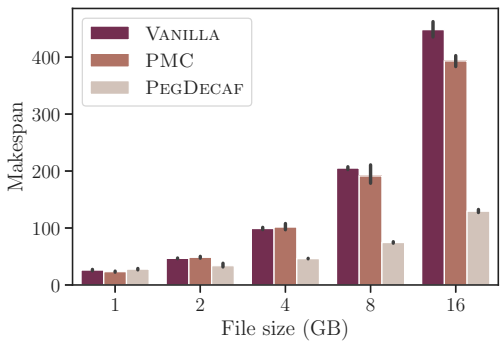
(b) Normalized makespan over VANILLA

Figure 5: Makespan and normalized makespan of SYNTHETICIO without sleep.

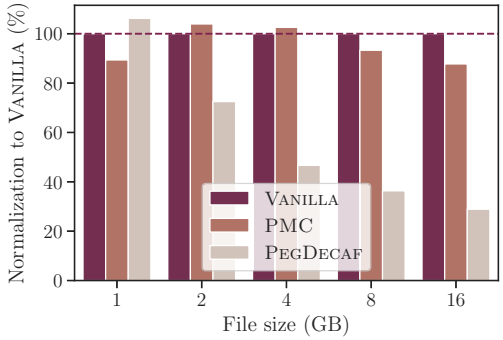
output files, which are given as input files to the child jobs. One benefit of using Decaf and in situ communications is that instead of performing I/O operations (e.g., writing files) after computing, Decaf can start sending data using asynchronous MPI calls, thus overlapping some parts of the computation with communications.

GENOME. The analysis presented in this section is based on the execution of a subset of jobs (*Individuals* and *Individuals_Merge*) of the GENOME workflow (see Section IV-B). In this experiment, we set the number of chromosomes processed by the workflow to 1, and we vary the number of *Individuals* jobs between 1 and 16. Note that increasing the number of *Individuals* jobs reduces the amount of data processed per job, either using files (for VANILLA and PMC) or in-memory transfers (for PEGDECAF). As in the previous experiment, we present the makespan and the normalized makespan, and each reported result is the average over 5 runs.

Figures 7a and 7b present the makespan of the GENOME workflow for PEGDECAF compared with the other two scenarios VANILLA and PMC. Similarly to the SYNTHETICIO workflow, the difference between VANILLA and PMC is due to the job submission mode. In VANILLA scenario, we submit several jobs to the scheduler, while in PMC we submit only one job that is managed by the PMC engine. The higher the number of *Individuals* jobs, the larger the variation in



(a) Makespan



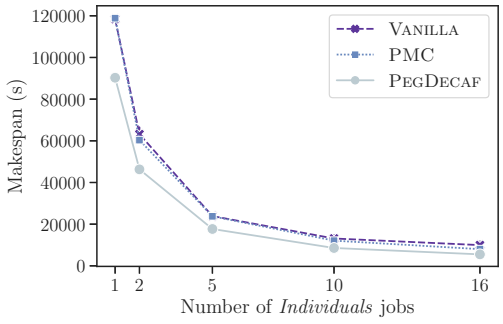
(b) Normalized makespan over Vanilla

Figure 6: Makespan and normalized makespan of SyntheticIO with sleep (2 seconds per GB).

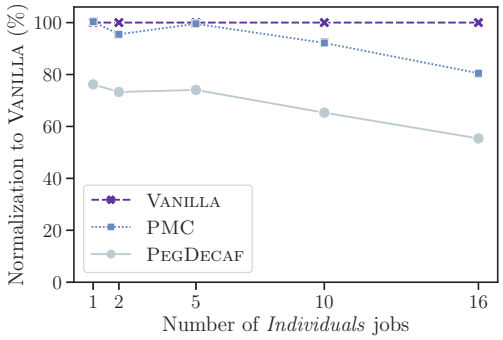
the execution time, which exacerbates the difference between the Vanilla and the PMC. Recall that the makespan in the Vanilla scenario is measured by the maximum execution time among *Individuals* jobs plus the time to perform *Individuals_Merge* (see Table III). The best overall makespan is achieved by PegDecaf, which illustrates the advantage of using in situ communications between jobs rather than traditional file-based I/Os. By normalizing to Vanilla, we see that the performance improvement brought by PegDecaf increases with the higher number of *Individuals* jobs (i.e., by 22% to 30% improvements in the makespan). The reason is that there are more concurrent communications with a higher number of *Individuals* jobs, which enhances the performance of in situ communications between jobs with MPI.

Figure 7c presents a strong scaling study. We study the speedup of Genome when varying the number of *Individuals* jobs between 1 and 16. The problem size is set to one chromosome (i.e, $250,000$ lines to process for all *Individuals*). The speedup is computed as the ratio between the time taken with 1 *Individuals* job and the time taken with x *Individuals* jobs. Figure 7c confirms our previous findings, PegDecaf clearly outperforms both baseline Vanilla and PMC; furthermore, PegDecaf achieves near-optimal speedup.

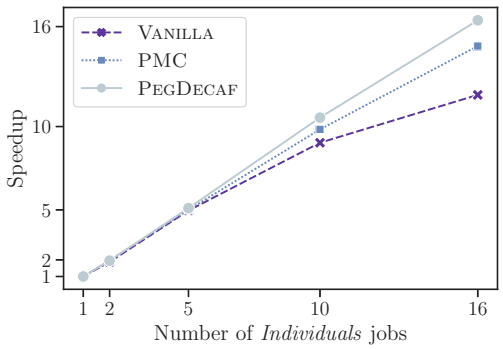
In Figure 8, we explore weak scaling. The problem size is now set per *Individuals* job, so each *Individuals* has to



(a) Makespan



(b) Normalized makespan over Vanilla

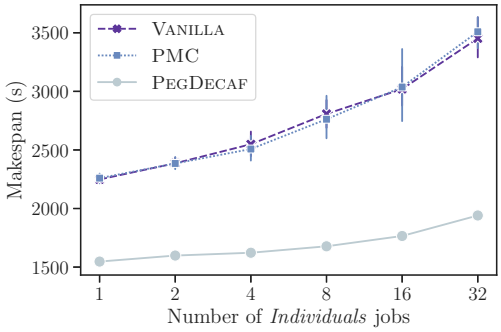


(c) Strong scaling speedup

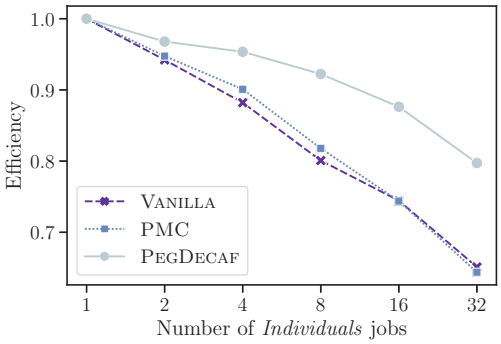
Figure 7: Makespan, normalized makespan and speedup of Genome with 1 chromosome when varying the number of *Individuals* jobs. The speedup is computed as $t(1)/t(x)$ where $t(x)$ is the makespan with x *Individuals* job (ideal speedup is x).

process $250,000/50 = 5,000$ lines and, each *Individuals* job runs on one dedicated node. These settings allow us to explore how PMC and PegDecaf scale up with the amount of data. Figures 8a and 8b respectively depicts the makespan and the efficiency of Genome when varying the number of *Individuals* jobs. Similarly, to the speedup in the strong scaling case, the efficiency is computed as the ratio between the time taken with 1 *Individuals* and the time taken

with x *Individuals* jobs. Figure 8a confirms that PEGDECAF executes the sub-workflow much faster than PMC. Figure 8b shows that both engines exhibit degradation when scaling up (recall that 1 is the perfect efficiency), however, PEGDECAF clearly outperforms PMC in terms of efficiency. The gap between PMC and PEGDECAF increases with the number of *Individuals* jobs, indicating that PEGDECAF scales better than PMC. We conjecture that this gap will only get larger for an higher number of *Individuals* jobs.



(a) Weak scaling makespan



(b) Weak scaling efficiency

Figure 8: Weak scaling study of GENOME where each *Individuals* processes $5,000$ lines. The efficiency is computed as $t(1)/t(x)$ where $t(x)$ is the makespan with x *Individuals* job (ideal efficiency is 1).

Summary. This campaign of experiments highlights the benefits of integrating approaches developed by the HPC and the in situ community into more traditional WMSs. Our results showed that using in situ communications improves the makespan of data-intensive workflows like GENOME. Moreover, as demonstrated with SYNTHETICIO, in situ communications can also improve the execution time of simpler workflows, and such improvements can even be larger when there is a room for overlapping communications with computations.

We also show that the benefits brought by using in situ frameworks in WMSs highly depend on the type of workflows. For instance, we suspect that scientific workflows may not benefit from such integration if there is an imbalance between computation and communication (i.e., the communication part is not large enough compared with the computation part). Therefore, analyzing the workflow characteristics (i.e., computation and communication behaviours) is key in finding appropriate scientific workflows that would benefit from in situ workflow engines.

Finally, another advantage of using Decaf is to reduce performance variations among different runs. Indeed, MPI communications are faster, but also much more stable than file-based I/Os, which rely on a parallel filesystem, a very complex piece of software subject to important performance variations.

VI. CONCLUSION

Scientific workflows have become one of the pillars of modern computational science. However, they still mostly rely on file-based communications and lack support for emerging approaches such as in situ. This study addresses the gap between traditional file communications in WMSs and recent HPC communication paradigms, where we have proposed a new workflow execution engine in Pegasus based on the in situ framework Decaf. This new engine improves the performance of scientific workflows by replacing costly file-based communications with faster MPI communications managed by Decaf. Using a realistic bioinformatics workflow, we have demonstrated that executing data-intensive jobs using Decaf reduces the execution time by 22% to 30%. In addition, from a user's perspective, our solution requires minimal changes in the existing Pegasus workflows, therefore, users can easily transition from using existing Pegasus engines to this new approach.

As future work, we will consider expanding experiments to use cases from other scientific domains to confirm these promising findings. We also plan to increase the scale of the experiments to further highlight the gap between MPI- and file-based communications. In this work, we choose which jobs should be managed by a specialized engine based on our knowledge of the workflow characteristics, a longer-term future work will be to investigate automated methods to identify these sub-workflows.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, program manager Margaret Lentz, under contracts number DE-AC02-06CH11357 and DE-AC02-05CH11231, and program manager Richard Carlson, under contracts number #DE-SC0012636 and #DE-SC0022328. This work is also supported by the National Science Foundation under contract number #1664162.

REFERENCES

[1] E. Huerta, R. Haas, S. Jha, M. Neubauer, and D. S. Katz, "Supporting High-Performance and High-Throughput Computing for Experimental science," *Computing and Software for Big Science*, vol. 3, no. 1, p. 5, 2019.

[2] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny, "The Evolution of the Pegasus Workflow Management Software," *Computing in Science & Engineering*, vol. 21, no. 4, pp. 22–36, 2019.

[3] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a Workflow Management System for Science Automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015, funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575. [Online]. Available: http://pegasus.isi.edu/publications/2014/2014-fgcs-deelman.pdf

[4] R. Mitchell, L. Pottier, S. Jacobs, R. Ferreira da Silva, M. Rynge, K. Vahi, and E. Deelman, "Exploration of Workflow Management Systems Emerging Features from Users Perspectives," in *First International Workshop on Big Data Tools, Methods, and Use Cases for Innovative Scientific Discovery (BTSD)*, 2019, pp. 4537–4544, funding Acknowledgments: NSF 1842042.

[5] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, and K. Vahi, "CyberShake: A physics-based seismic hazard model for southern California," *Pure and Applied Geophysics*, vol. 168, no. 3, pp. 367–381, 2011.

[6] E. Huerta, A. Khan, X. Huang, M. Tian, M. Levental, R. Chard, W. Wei, M. Heflin, D. S. Katz, V. Kindratenko, D. Mu, B. Blaiszik, and I. Foster, "Accelerated, Scalable and Reproducible AI-driven Gravitational Wave Detection," *Nature Astronomy*, vol. 5, no. 10, pp. 1062–1068, 2021.

[7] J. S. Vetter, R. Brightwell, M. Gokhale, P. McCormick, R. Ross, J. Shalf, K. Antypas, D. Donofrio, T. Humble, C. Schuman, B. Van Essen, S. Yoo, A. Aiken, D. Bernholdt, S. Byna, K. Cameron, F. Cappello, B. Chapman, A. Chien, M. Hall, R. Hartman-Baker, Z. Lan, M. Lang, J. Leidel, S. Li, R. Lucas, J. Mellor-Crummey, P. Peltz Jr., T. Peterka, M. Strout, and J. Wilke, "Extreme Heterogeneity 2018 - Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 12 2018.

[8] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, "Data Movement Is All You Need: A Case Study on Optimizing Transformers," in *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica, Eds., vol. 3, 2021, pp. 711–732. [Online]. Available: https://proceedings.mlsys.org/paper/2021/file/c9e1074f5b3f9fc8ea15d152add07294-Paper.pdf

[9] D. Raften, "System-managed CF Structure Duplexing," *IBM e-server zSeries*, 2004.

[10] C. S. Daley, D. Ghoshal, G. K. Lockwood, S. Dosanjh, L. Ramakrishnan, and N. J. Wright, "Performance Characterization of Scientific Workflows for the Optimal Use of Burst Buffers," *Future Generation Computer Systems*, vol. 110, pp. 468–480, 2020.

[11] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, and E. Bethel, "In situ methods, infrastructures, and applications on high performance computing platforms," in *Computer Graphics Forum*, vol. 35, no. 3. Wiley Online Library, 2016, pp. 577–597.

[12] M. Dorier, M. Dreher, T. Peterka, J. M. Wozniak, G. Antoniu, and B. Raffin, "Lessons Learned from Building In Situ Coupling Frameworks," in *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, 2015, pp. 19–24.

[13] M. Dreher and T. Peterka, "Decaf: Decoupled dataflows for in situ high-performance workflows," Argonne National Lab.(ANL), Argonne, IL (United States), Tech. Rep., 2017.

[14] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi, "Enabling In-situ Execution of Coupled Scientific Workflow on Multi-core Platform," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 1352–1363.

[15] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A Survey of Data-intensive Scientific Workflow Management," *Journal of Grid Computing*, vol. 13, no. 4, pp. 457–493, 2015.

[16] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A Characterization of Workflow Management Systems for Extreme-scale Applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, 2017.

[17] B. Whitlock, J. M. Favre, and J. S. Meredith, "Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System," in *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, ser. EGPGV '11. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2011, pp. 101–109. [Online]. Available: http://dx.doi.org/10.2312/EGPGV/EGPGV11/101-109

[18] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, and J. Mauldin, "ParaView Catalyst: Enabling in situ data analysis and visualization," in *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. ACM, 2015, pp. 25–29.

[19] M. Dorier, G. Antoniu, F. Cappello, M. Snir, R. Sisneros, O. Yildiz, S. Ibrahim, T. Peterka, and L. Orf, "Damaris: Addressing Performance Variability in Data Management for Post-petascale Simulations," *ACM Transactions on Parallel Computing (TOPC)*, vol. 3, no. 3, pp. 1–43, 2016.

[20] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: An Interaction and Coordination Framework for Coupled Simulation Workflows," *Cluster Computing*, vol. 15, no. 2, pp. 163–181, 2012.

[21] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki, "Flexpath: Type-based Publish/Subscribe System for Large-scale Science Analytics," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2014, pp. 246–255.

[22] O. Yildiz, J. Ejarque, H. Chan, S. Sankaranarayanan, R. M. Badia, and T. Peterka, "Heterogeneous Hierarchical Workflow Composition," *Computing in Science & Engineering*, vol. 21, no. 4, pp. 76–86, 2019.

[23] E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R. M. Badia, J. Torres, T. Cortes, and J. Labarta, "PyCOMPSs: Parallel computational workflows in Python," *The International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 66–82, 2017.

[24] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: the Condor Experience," *Concurrency and computation: practice and experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[25] G. Singh, M.-H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta, "Workflow task clustering for best effort systems with Pegasus," in *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*, 2008, pp. 1–8.

[26] M. Rynge, G. Juve, K. Vahi, S. Callaghan, G. Mehta, P. J. Maechling, and E. Deelman, "Enabling Large-scale Scientific Workflows on Petascale Resources Using MPI Master/Worker," in *XSEDE12*, 2012, funding acknowledgements: NSF OCI-0722019, NSF OCI-0943725, NSF EAR-0529922, USGS 07HQAG0008, NSF OCI-1053575. [Online]. Available: http://pegasus.isi.edu/publications/2012/XSEDE12-Rynge-pegasus-mpi-cluster.pdf

[27] D. W. Walker and J. J. Dongarra, "MPI: A Standard Message Passing Interface," *Supercomputer*, vol. 12, pp. 56–68, 1996.

[28] M. Dreher and T. Peterka, "Bredala: Semantic Data Redistribution for In Situ Applications," in *Proceedings of IEEE Cluster 2016*. IEEE, 2016.

[29] "Pegasus I/O Synthetic Workflow (GitHub)," https://github.com/pegasus-isi/io-synthetic.

[30] "Pegasus Workflow 1000Genome (GitHub)," https://github.com/pegasus-isi/1000genome-workflow.

[31] . G. P. Consortium, "A Global Reference for Human Genetic Variation," *Nature*, vol. 526, no. 7571, p. 68, 2015.

[32] "NERSC, Lawrence Berkeley National Laboratory's Supercomputer Cori," https://www.nersc.gov/users/computational-systems/cori.

[33] "Pegasus WMS (GitHub)," https://github.com/pegasus-isi/pegasus.

[34] "Decaf (GitHub)," https://github.com/tpeterka/decaf.

[35] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple Linux Utility for Resource Management," in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003, pp. 44–60.